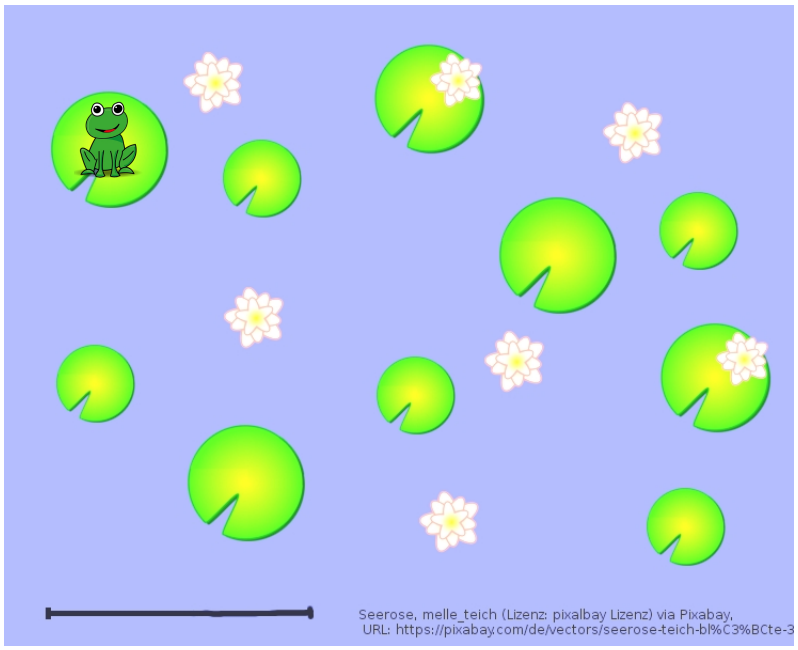
Problemstellung

Quacki sitzt in seinem wunderschönen Seerosenteich gemütlich auf einem Seerosenblatt und wartet auf Fliegen - seine Lieblingsspeise. Gelegentlich hüpft er zu einem anderen Blatt, wenn dort eine Fliege umherfliegt. Um möglichst schnell bei der Fliege zu sein, nimmt er denjenigen Weg, bei dem er am wenigsten Sprünge machen muss. Wie viele Sprünge muss Quacki maximal machen, um auf diese Weise zu einem beliebigen anderen Blatt zu kommen.



Seerosenteich, melle_teich (Lizenz: pixabay Lizenz) via Pixabay,
URL: https://https://pixabay.com/de/vectors/seerose-teich-bl%C3%BCte-3347654/

Hilf ihm bei der Antwort! Dabei muss die maximale Sprungweite (schwarzer Balken im Bild) berücksichtigt werden. Die Entfernung wird immer von Mitte zur Mitte der Blätter gemessen.

1. *Bestimme die Anzahl der notwendigen Sprünge zu jedem anderen Blatt.*

Modellierung

Die Ausgangssituation soll nun als Graph modelliert werden.

2. *Entscheide, welche der folgenden Informationen wichtig für die Bestimmung sind:*
   - Anzahl der Blätter im Teich
   - Größe des Teichs
   - Größe der Blätter
   - exakte Entfernung der Blätter
   - exakte Position der Blätter
   - liegt der Abstand der Blätter unter der maximalen Sprungweite

Modellierung

Knoten:

Kanten:

Kürzeste Entfernung in ungerichteten Graphen

Weiterführende Fragen

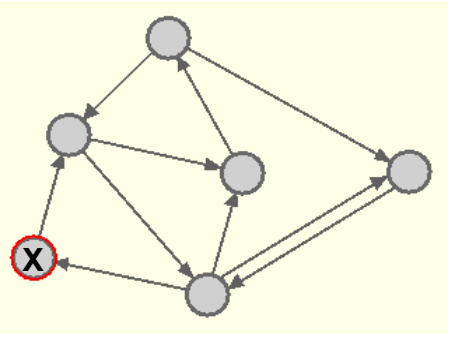
Aufgaben

3. *Untersuche, was passiert, wenn man statt Breitensuche Tiefensuche bei dem Algorithmus verwendet.*
4. *Entwirf eine Anpassung des Algorithmus, die mit Tiefensuche arbeitet und trotzdem die richtige Entfernungen bestimmt.*

Für viele Anwendungen verwendet man gerichtete Graphen, d.h. die Kanten haben eine Richtung (z.B. bei einem Stadtplan mit Einbahnstraßen). Die Kanten dürfen dann nur in der vorgegebenen Richtung durchlaufen werden.

Aufgaben

5. *Bestimme die Entfernungen in nebenstehendem Graphen vom markierten Knoten aus.*
6. *Entscheide, ob der Algorithmus verändert werden muss, um auf gerichtete Graphen angewendet werden zu können.*



gerichteter Graph (eigenes Werk)

Gestufte Hilfen für selbstentdeckendes Verfahren

Hilfe 1:

Wie ergibt sich die Anzahl der notwendigen Sprünge zu einem Blatt, wenn man weiß, dass man zu einem benachbarten Blatt fünf Sprünge (n Sprünge) braucht? Formuliere diese Aussage auch mit Knoten und Kanten.

Hilfe 2:

Die Entfernungen welcher Knoten kann man am Anfang des Algorithmus sehr leicht bestimmen?

Hilfe 3:

Man weiß von einem Knoten, dass er die Entfernung 2 vom Startknoten hat, und von einem anderen, dass er die Entfernung 1 hat. Nun sollen die Entfernungen zu den Nachbarn dieser beiden Knoten bestimmt werden. Welchen Knoten würdest du zuerst auswerten?

Hilfe 4:

Lässt sich die Reihenfolge der Abarbeitung besser durch Breiten- oder durch Tiefensuche realisieren?

Beschreibung des Algorithmus A von Edward Moore

Um das Problem des kürzesten Pfades in einem ungewichteten Graphen zu lösen, verwendete E. Moore Breitensuche: Er verwendet eine Schlange (Queue), um die Liste der noch zu bearbeitenden Knoten zu speichern. Dort wird am Anfang der Startknoten eingefügt. Dieser erhält die Entfernung 0 und wird als besucht gekennzeichnet.

Dann wiederholt man folgende Schritte bis die ToDo-Liste leer ist: Der vorderste Knoten wird aus der Liste entfernt. Er ist der aktuelle Knoten. Er wird als fertig bearbeitet markiert. Dann untersucht man alle seine Nachbarknoten. Sind diese noch nie besucht worden, dann trägt man bei ihnen die Entfernung des aktuellen Knotens plus 1 ein, kennzeichnet sie als besucht und fügt sie am Ende der ToDo-Liste ein.

Am Ende sind alle erreichbaren Knoten als fertig bearbeitet markiert und ihre Entfernung vom Startknoten eingetragen. Nicht erreichbare Knoten sind noch nicht als besucht gekennzeichnet und haben keine Entfernung eingetragen.

Pseudocode des Algorithmus A von Edward Moore

Der hier beschriebene Algorithmus führt eine Breitensuche durch. Knoten, die am Ende nicht als besucht markiert sind, sind nicht mit dem Startknoten verbunden und es kann keine Entfernung bestimmt werden.

Algorithmus A:

```
Setze den Entfernungswert des Startknotens auf 0
Erzeuge eine leere ToDo-Liste und füge den Startknoten hinzu

Wiederhole solange die ToDo-Liste nicht leer ist
  Nimm den ersten Knoten aus der ToDo-Liste heraus
  Sein Entfernungswert ist d
  Markiere ihn
  Wiederhole für jeden seiner Nachbarn
    Falls der Nachbarknoten noch nicht besucht ist
      Kennzeichne ihn als "besucht"
      Setze den Entfernungswert auf d+1
      Füge ihn am Ende der ToDo-Liste hinzu
    Ende-Falls
  Ende-Wiederhole
Ende-Wiederhole
```

Quelltext des Algorithmus A von Edward Moore

```
public boolean istZusammenhaengend() {
    if (g.getAnzahlKnoten()==0) {
        return;
    }

    ArrayList<Knoten> toDo = new ArrayList<Knoten>();
    getStartKnoten().setBesucht(true);
    getStartKnoten().setWert(0);
    toDo.add(getStartKnoten());

    while(toDo.size()>0) {
        Knoten k = toDo.remove(0);
        k.setMarkiert(true);
        for(Knoten n : g.getNachbarKnoten(k)) {
            if(!n.isBesucht()){
                n.setWert(k.getIntWert()+1);
                toDo.add(n);
                g.getKante(k,n).setMarkiert(true);
                n.setBesucht(true);
            }
        }
    }
```